滑动均值聚类*

何沧平
(微博, cangping@staff.weibo.com, cphe@lsec.cc.ac.cn)
孟令霞
(北京第二实验小学通州分校, menglingxiahbcz@gmail.com)

摘　要

本文提出一个名为滑动均值的聚类算法，尝试替代常用的k均值算法。滑动均值能处理大量的样本，自行决定类别数量，用混洗样本来避免出现很差的中心点，能够中途裁减类别数量，聚类效果显著好于k均值。在鸢尾花数据和手写数字数据上，滑动均值的聚类效果比k均值分别高9.93%和5.17%。

关键词: 滑动均值，k均值，聚类

Sliding Means Clustering

He Cangping
(*WEIBO.COM, cangping@staff.weibo.com, cphe@lsec.cc.ac.cn*)
Meng Lingxia
(*Tongzhou branch of No.2 Experimental Primary School, menglingxiahbcz@gmail.com*)

Abstract

This paper proposes a novel clustring algorithm named *Sliding Means*, aiming to take the place of k-means algorithm which is widely used in internet applications. Sliding means has the ability to handle with very large datasets, and to automatically determine the number of clusters. With the help of shuffling samples, bad initial centroids have little chance to be selected. Sliding means is also able to drop some bad centroids on the fly. On the iris dataset and optdigits dataset, sliding means achieves better performance(Adjusted Rand Index) than k-means by 9.93% and 5.17% respectively.

Keywords: sliding means, k-means, clustering

1. 引言

随着深度学习的快速发展，互联网业务中的推荐系统、计算机视觉、自然语言处理都广泛使用嵌入技术，图片、视频、文本、用户、物品都逐渐转换成了向量，相互之间可以方便地计算距离，从而聚类需求多了起来。例如，最近天猫披露其推荐系统通过对最近几天的商品向量聚类 [2]，来获取用户的多个兴趣向量。

互网联应用中，用户数量和待聚类的样本数量通常很大，动辄上亿，这就要求聚类算法能够处理很多的样本。scikit-learn手册[1)]中比较了常用的10种聚类算法，能够处理很多样本的算法有k均值 [3]、DBSCAN [4]和OPTICS [5]。这3种算法中，k均值的使用相对更广泛。但k均值有个缺点：性能依赖于k个初始中心点的质量。因此，中心点的初始化方法成为提高性能的关键。

* 2019年11月26日提交.

1) https://scikit-learn.org/stable/modules/clustering.html#adjusted-rand-index

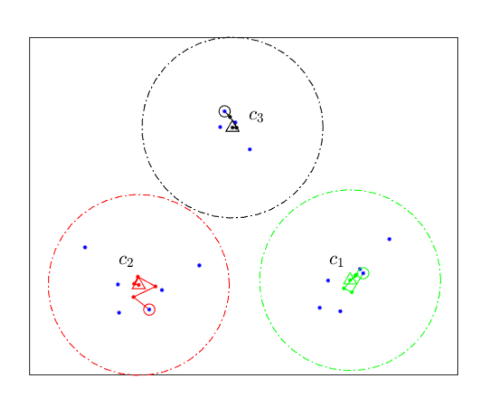


图 1: 滑动均值的中心点更新轨迹。蓝点的是样本，小圆圈内的蓝点是起始中心点，小三角形最终得到的中心点c_1, c_2, c_3。3个虚线大圆是以3个中心点为圆心、以指定距离为半径绘制。

本文从解决初值依赖性出发，设计了一个滑动均值算法，采用一些新的技巧：将类别数量由人为指定改为算法自主决定、混洗样本以避免选出很差的中心点、中途裁减中心点数量，从而实现能够处理比k均值更多的样本，达到更高的性能。在鸢尾花数据和手写数字数据上，滑动均值的聚类效果(Adjusted Rand Index [11])比k均值分别高9.93%和5.17%。

本文后续内容这样组织。第2节是相关工作，第3节给出滑动均值的具体步骤，第4节实验给出鸢尾花数据和手写数字数据集上的测试结果，第5节总结合文。

2. 相关工作

聚类是机器学习的基础课题，机器学习教科书 [1]都会讲解若干种成熟的聚类方法，例如k均值聚类、层次聚类、密度聚类，这里不赘述。在MATLAB1) 、scikit-learn2) 、scipy3) 等机器学习软件中，k均值的初始化方法默认选用K-means++ [6]，备选项还有随机选取k个样本作为中心点、按照均匀分布或者高斯分布随机选取中心点。K-means++ 也是随机选取中心点，但会采取措施使中心点尽量相互远离，在一定程度上解决初值依赖性。Canopy聚类 [7]不必指定簇的数量，适合并行计算。MIND [2]改造胶囊网络，用于商品向量聚类。

3. 滑动均值

N个样本记为$x_1, x_2, \ldots, x_N$，每个样本x_i都是一个向量。滑动均聚类的任务是将所有样本分成若干类，使同类的样本相互靠近，不同类的样本相互远离。

滑动均值聚过程为：生成若干初始中心点→ 迭代若干轮直至中心点达到稳定状态→ 根据聚类效果调整中心点数量→ 如果中心点过少，那么重新生成中心点，如果中心点过多，那么裁减中心点。

生成初始中心点。直观上来看，中心点周围应该聚焦一些样本点。选取样本x_1作为第一个中心点，记为c_1。再指定一个半径σ_0。按下标顺序检查每一个样本$x_i, i = 1, 2, \ldots, N$，如果x_i与中心点c_1的距离小于等于半径σ_0，那么将样本x_i标记为第1类，并用x_i来更新中心点c_1，使中心点c_1向样本x_i滑动一点点；如果样本x_i与中心点c_1的距离大于半径σ_0，那么将x_i做为一个新的中心点c_2，并将x_i标记为第2类。假设已经生成了K个中心点$c_1, c_2, \ldots, c_K$。计算样本x_i 与中心点$c_1, c_2, \ldots, c_K$之

1) https://ww2.mathworks.cn/help/stats/kmeans.html?requestedDomain=cn
2) https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans
3) https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.vq.kmeans2.html#scipy.cluster.vq.kmeans2

间的距离，如果x_i落在至少1个中心点的半径之内，那么找出与样本x_i距离最近的中心点c_k，然后将样本x_i标记为第k类，并用x_i来更新中心点c_k，使中心点c_k向样本x_i滑动一点；如果样本x_i在所有中心点的半径之外，那么将x_i做为一个新的中心点c_{K+1}，并将x_i标记为第$K+1$类。这样进行下去，直至将所有的样本检查一遍。严谨的步骤见算法1。

图1可以直观说明生成初始中心点的过程。肉眼观察，所有18个样本应当聚为3类，3类样本分别位于3个虚线大圆内。以红色大圆内的样本为例，中心点c_2的取值最开始是红色小圆内的样本，随着检查的进行，c_2按着红色折线一步步地滑动到红色小三角形内。

算法1中，不直接指定半径σ_0，而指定一个系数r，通过r乘以样本方差σ_0间接到半径，这样做可以在一定程度降低猜测半径的难度。中心点数量K不用人为指定，算法会自动决定其取值。第3行的函数dist$(\cdot,\cdot)$代表两点之间的距离，可以是欧式距离、余弦距离或者别的距离。第4~9行新增一个中心点。第10~12行是中心点的滑动操作：第10行选出x_i归属的中心点$c_{\hat{k}}$，第11行将中心点$c_{\hat{k}}$上的样本数量加1，第12行将样本x_i按权重加到中心点$c_{\hat{k}}$上，完成滑动。第12行公式中的权重系数$\frac{\hat{k}-1}{\hat{k}}$和$\frac{1}{\hat{k}}$保证滑动更新得到的中心点恰好是该类样本的均值。

算法 1: init$(\cdot)$，生成初始中心点

Input: 正实数r，N个样本$x_1,x_2,\ldots,x_N$。
Output: 中心点数量K，中心点$c_1,c_2,\ldots,c_K$，各类包含的样本数量$n_1,n_2,\ldots,n_K$。

1 令$\mu=\frac{1}{N}\sum_{i=1}^{N}x_i$, $\sigma^2=\frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2$, $\sigma_0^2=r^2\sigma^2$, $n_1=0$, $K=1$, 中心点$c_1=x_1$。
2 **for** i=1:N **do**
3 　计算距离$d_{ik}=\mathrm{dist}(x_i,c_k), k=1,2,\ldots,K$。
4 　**if** 对$\forall k=1,2,\ldots,K$均有$d_{ik}^2>\sigma_0^2$ **then**
5 　　$K=K+1$
6 　　增加一个中心点$c_K=x_i$。
7 　　$n_K=0$
8 　　$d_{ik}=\mathrm{dist}(x_i,c_k), k=1,2,\ldots,K$
9 　**end**
10 　$\hat{k}=\arg\min_{k=1}^{K}(d_{ik})$
11 　$n_{\hat{k}}=n_{\hat{k}}+1$
12 　$c_{\hat{k}}=\frac{\hat{k}-1}{\hat{k}}c_{\hat{k}}+\frac{1}{\hat{k}}x_i$
13 **end**

算法 2: trian_oneepoch$(\cdot)$，进行一轮训练

Input: 中心点数量K，中心点$c_1,c_2,\ldots,c_K$，各类中样本的数量$n_1,n_2,\ldots,n_K$。
Output: 中心点$c_1,c_2,\ldots,c_K$，各类中样本的数量$n_1,n_2,\ldots,n_K$。

1 对$k=1,2,\ldots,K$，令$c_k^o=c_k$, $n_k^o=n_k$, $n_k=0$。
2 **for** i=1:N **do**
3 　计算距离$d_{ik}=\mathrm{dist}(x_i,c_k), k=1,2,\ldots,K$。
4 　$\hat{k}=\arg\min_{k=1}^{K}(d_{ik})$
5 　$n_{\hat{k}}=n_{\hat{k}}+1$
6 　**if** $n_k\le n_{\hat{k}}^o$ **then**
7 　　$c_{\hat{k}}=c_{\hat{k}}+\frac{1}{n_{\hat{k}}^0}(x_i-c_{\hat{k}}^o)$
8 　**else**
9 　　$c_{\hat{k}}=\frac{n_{\hat{k}}-1}{n_{\hat{k}}}c_{\hat{k}}+\frac{1}{n_{\hat{k}}}x_i$
10 　**end**
11 **end**
12 **for** $k=1:K$ **do**
13 　**if** $n_k<n_k^o$且$n_k\ge 1$ **then**
14 　　$c_k=\frac{n_k^o}{n_k}c_k-\frac{n_k^o-n_k}{n_k}c_k^o$
15 　**end**
16 **end**

算法1生成的中心点，可能还不稳定，需要再训练几轮，一轮训练过程见算法2。算法2的输入是算法1中得到的中心点数量K、各类的中心点$c_1,c_2,\ldots,c_K$和各类中样本的数量$n_1,n_2,\ldots,n_K$，输出是滑动更新后的中心点和各类样本的数量。算法2中，第1行将c_k^o和n_k^o作为旧值保存下来，以便在后续更新c_k和n_k时使用。第5行对样本计数，一直累加，当本轮的样本数量n_k小于等于前轮样本数量n_k^o时，用第7行的公式使中心点向当前样本滑动一点点；当本轮的样本数量n_k大于前轮样本数量n_k^o时，用第9行的公式使中心点向当前样本滑动一点点。在所有样本都参与滑动以后，如果本轮数量n_k小于前轮数量n_k^o时，用第14行的公式进行最后一次滑动，此时旧中心点c_k^o参与了计算。第7、9、12行的三个公式能够保证滑动更新得到的中心点恰好是该类样本的均值。

用算法2迭代若干轮达到稳定状态以后，再根据聚类效果来决定接下来是增加中心点数量还是减少中心点数量。如果觉得中心点数量不够多，那么需要将实数r调小一些，从算法1重新开始。如果觉得中心点太多了，那么将实数r调大一些，从算法1重新开始。如果中心点数量多得不太多，那么用算法3裁减掉一些。

裁减中心点。算法3的输入参数中，中心点$c_1, c_2, \ldots, c_K$和样本数量$n_1, n_2, \ldots, n_K$都是算法2的输出，类别数量上限为$\hat{K}$需要手动指定，显然有$\hat{K} < K$。输出是$\hat{K}$个中心点$c_1, c_2, \ldots, c_{\hat{K}}$，$\hat{K}$个样本数量$n_1, n_2, \ldots, n_{\hat{K}}$。第1~8行删除那些没有样本归附的中心点。第10~14行去掉样本数量最少的那个中心点，然后进行若干轮训练，当新旧中心点的相对距离小于ϵ时（第18行），停止训练。

综合本节的描述，就能得到滑动均值聚类的全过程，具体写出来就算法5。算法5中不显眼但重要的操作是混洗样本，第1行的混洗，使得初始中心点具有随机性，避免陷入某个局部最优解。第4行的混洗能显著提高聚类效果。特别注意，裁减中心点的算法3中没有混洗样本，否则会显著降低聚类效果。

批量训练。由算法5可知，实际计算过程中，生成初始中心点的函数init(·)只调用1次，而训练函数trian_oneepoch(·)会调用很多次，但trian_oneepoch(·)每步只处理一个样本，不能充分利用CPU的向量化计算能力、内存的按块读写能力。因此，将算法2改写算法4，每步处理$S \geq 1$个样本。样本与中心点之间的距离带来的计算量占算法2计算量的绝大部分，批量处理后，算法4的第3行比算法2的第3行的单步计算量增加S倍，函数dist(·,·)的调用次数减少至原来的$\frac{1}{S}$，大幅提高代码运行速度。

算法 3: cutcentroid(·)，裁减类中心

Input: 类别数量上限为$\hat{K}$，类别数量K，中心点$c_1, c_2, \ldots, c_K$，各类中样本的数量$n_1, n_2, \ldots, n_K$。

Output: 类中心$c_1, c_2, \ldots, c_{\hat{K}}$，各类中样本的数量$n_1, n_2, \ldots, n_{\hat{K}}$。

1 **for** k=1:K **do**
2 　**if** $n_k = 0$ **then**
3 　　去掉c_k和n_k。
4 　　$K = K - 1$ 。
5 　　剩余的中心点重新编号为$c_1, c_2, \ldots, c_K$。
6 　　剩余的样本数重量新编号为$n_1, n_2, \ldots, n_K$。
7 　**end**
8 **end**
9 **while** $\hat{K} < K$ **do**
10 　$k = \arg\min_{i=1}^{K}(n_i)$
11 　去掉c_k和n_k。
12 　$K = K - 1$。
13 　剩余的中心点重新号为$c_1, c_2, \ldots, c_K$。
14 　剩余的样本数量重新编号为$n_1, n_2, \ldots, n_K$。
15 　**for** $i = 1 : M$ **do**
16 　　令$c_k^o = c_k$, $k = 1, 2, \ldots, K$。
17 　　用trian_oneepoch(·)训练一轮。
18 　　**if** $\frac{\sum_{k=1}^{K} ||c_k - c_k^o||^2}{\sum_{k=1}^{K} ||c_k||^2 + \sum_{k=1}^{K} ||c_k^o||^2} < \epsilon^2$ **then**
19 　　　跳出for循环。
20 　　**end**
21 　**end**
22 **end**

算法 4: trian_oneepoch_batch(·)，一轮批量训练

Input: 样本$x_1, x_2, \ldots, x_N$，类别数量K，类中心$c_1, c_2, \ldots, c_K$，各类中样本的数量$n_1, n_2, \ldots, n_K$，正整数$S \geq 1$。

Output: 类中心$c_1, c_2, \ldots, c_K$，各类中样本的数量$n_1, n_2, \ldots, n_K$

1 对$k = 1, 2, \ldots, K$, 令$c_k^o = c_k$, $n_k^o = n_k$, $n_k = 0$
2 **for** i=1:S:N **do**
3 　$d_{jk} = \mathrm{dist}(x_j, c_k), k = 1, 2, \ldots, K, j = i, i+1, \ldots, \min(N, i+S-1)$ 。
4 　**for** $k = 1 : K$ **do**
5 　　令子集$B_k = \{x_j | d_{jk} = \min_{t=1}^{K}(d_{jt})\}$。
6 　　B^k中的元素数量记为n_k^b。
7 　　$n_k = n_k + n_k^b$
8 　　**if** $n_k^b > 0$且$n_k \leq n_k^o$ **then**
9 　　　$c_k = c_k - \frac{n_k^b}{n_k^o} c_k^o + \frac{1}{n_k^o} \sum_{x \in B_k} x$
10 　　**end**
11 　　**if** $n_k^b > 0$且$n_k - n_k^b \leq n_k^o < n_k$ **then**
12 　　　$c_k = \frac{n_k^o}{n_k} c_k - \frac{n_k^o - n_k + n_k^b}{n_k} c_k^o + \frac{1}{n_k} \sum_{x \in B_k} x$
13 　　**end**
14 　　**if** $n_k^b > 0$且$n_k^o < n_k - n_k^b$ **then**
15 　　　$c_k = \frac{n_k - n_k^b}{n_k} c_k + \frac{1}{n_k} \sum_{x \in B_k} x$
16 　　**end**
17 　**end**
18 **end**
19 **for** $k = 1 : K$ **do**
20 　**if** $n_k < n_k^o$且$n_k \geq 1$ **then**
21 　　$c_k = \frac{n_k^o}{n_k} c_k - \frac{n_k^o - n_k}{n_k} c_k^o$
22 　**end**
23 **end**

算法 5: slidingmeans(·)，滑动均值聚类全过程

Input: 样本$x_1, x_2, \ldots, x_N$，最大训练步数M，最大误差$\epsilon > 0$，类别数量$\hat{K}$。

Output: 中心点c_k和各类样本数量n_k，$k = 1, 2, \ldots, \hat{K}$。

1 混洗样本$x_1, x_2, \ldots, x_N$，按混洗后的顺序将样本仍然记为$x_1, x_2, \ldots, x_N$。
2 用init(·)进行初始化。
3 **for** $i = 1 : M$ **do**
4   混洗样本$x_1, x_2, \ldots, x_N$。
5   用trian_oneepoch(·)训练一轮。
6   **if** $\frac{\sum_{k=1}^{K} ||c_k - c_k^o||^2}{\sum_{k=1}^{K} ||c_k||^2 + \sum_{k=1}^{K} ||c_k^o||^2} < \epsilon^2$ **then**
7   跳出for循环。
8   **end**
9 **end**
10 **if** $K > \hat{K}$ **then**
11   用cutcentroid(·)裁减中心点。
12 **else**
13   调小r_0，从第2行开始重新计算。
14 **end**

4. 实验

滑动均值与k均值都属于均值聚类算法，因此可以用k值作为基准来衡量滑动均值的性能。

本节对比滑动均值与k均值在Fisher的鸢尾花(iris)数据集 [8]和手写数字(optdigits)数据集 [10]上的性能。鸢尾花数据集可从 [9]获得，它包含了150个鸢尾花样本，每个样本有4个特征：花萼长度(sepal length)、花萼宽度(sepal width)、花瓣长度(petal length)、花瓣宽度(petal width)。同时标明了样本来自三种鸢尾属植物中的哪一种：Iris Setosa、Iris Versicolour、Iris Virginica，每种50个样本。手写数字数据集可从 [9]获得，它的训练集和测试分别包含3823和1797个样本，本节使用的是它的训练集。每个样本包含64个0~16的整数，代表8x8的位图，样本同时标明了属于0~9之间的哪个数字。

样本与中心点之间的距离有多种，这里选择欧式距离和余弦距离两种。k均值采用MATLAB R2019b自带的函数kmeans和scikit-learn 0.21.3自带的函数KMeans，kmeans同时支持欧式距离和余弦距离，KMeans只支持欧式距离。k均值的初始化方法选用2种：kmeans ++、随机选择k个样本做为初始中心点。

聚类性能评价指标使用Adjusted Rand Index (ARI) [11]。ARI是一个实数，若按均匀分布随机给样本指定类别，那么ARI接近0.0。如果聚类结果与真实标签相接近，那么ARI为正，如果聚类结果与真实标签完全相同，那么ARI为1.0.

每种算法配置下运行10000次，滑动均值的ARI和k均值的ARI列在了表1。表中第1~4对应鸢尾花数据集，第5~8行对应手写数字数据集。第1列和第5列中的sc代表滑动均值，km代表k均值，cos代表余弦距离，euc代表欧式距离，plus代表k-means++初始化方法，rand和sample代表随机选取初始中心点，matlab代表MATLAB自带函数kmeans，sklearn代表scikit-learn自带的函数KMeans。

对鸢尾花数据集，将10000运行得到的ARI从小到大排序，其变化走势在图2。从表1中可以看出，在鸢尾花余弦距离配置下，滑动均值的平均ARI是0.9094，比k均值的0.8161高0.0993；从图2(a-c)可知，滑动均值的ARI小于0.9039的比例只占1.75%，而k均值的ARI小于0.9039的比例分别占19.33%和24.69%。在鸢尾花欧式距离配置下，MATLAB版k均值的ARI明显低scikit-learn版，滑动均值的ARI平均值是0.7254，比scikit-learn版k均值小0.0048。从图2(d-h)可以看出，MATLAB的kmeans的ARI均值是0.7163，小于0.7162的比列分别为8.67%和21.00%；scikit-learn的KMeans的ARI最小值是0.7163，

表 1: 滑动均值与k均值的ARI比较

鸢尾花上的算法配置	平均	最小	最大	手写数字上的算法配置	平均	最小	最大
sc_cos	**0.9094**	**0.886**	**0.9222**	sc_cos	**0.6648**	**0.4969**	**0.782**
km_cosplus_matlab	0.8161	0.4394	0.9039	km_cosplus_matlab	0.627	0.4454	0.7756
km_cossample_matlab	0.7907	0.4394	0.9039	km_cossample_matlab	0.6241	0.4204	0.7754
sc_euc	0.7254	0.6898	**0.8176**	sc_euc	**0.7299**	0.5691	**0.7872**
km_eucplus_matlab	0.6975	0.4197	0.7302	km_eucplus_matlab	0.6312	0.4428	0.7789
km_eucsample_matlab	0.6614	0.4197	0.7302	km_eucsample_matlab	0.6289	0.4685	0.776
km_eucplus_sklearn	**0.7302**	**0.7163**	0.7302	km_eucplus_sklearn	0.6782	**0.6**	0.6883
km_eucrand_sklearn	**0.7302**	**0.7163**	0.7302	km_eucrand_sklearn	0.6765	0.581	0.7765

等于0.7163的比例分别是0.31%和0.61%。

概括来说，鸢尾花数据集聚类，应该选用余弦距离，滑动均值的ARI比k均值高9.93%。

对手写数字数据集，将10000运行得到的ARI作成直方图，见图3，横轴是ARI，纵轴是落在对应小区间的数字个数. 图3的每张图片中都显示出几个接近正态分布的形状，说明10000次运行已经能够反映滑动均值和k均值性能的全貌。观察表1的第5-8列，在手写数字余弦距离配置下，滑动均值的平均ARI是0.6648，高于k均值的0.627。在手写数字欧式距离配置下，滑动均值的平均ARI是0.7299，比k均值的ARI 0.6782高0.0517。仔细对比图3(dg)发现，滑动均值的ARI有53.37%的比例超过0.75，而k均值的ARI均小于0.75.

概括来说，手写数字数据集聚类，应该选用欧式距离，滑动均值的ARI比k均值高5.17%。

5. 总结与讨论

滑动均值有以下几个优点。

内存需求少。聚类过程中，只需要保存K个中心点向量、K个旧中心点向量和$O(K)$个标量，内存需求量比k均值持平。

计算量小。对滑动均值和k均值来说，每一轮训练的绝大部分计算量来自于中心点与样本间的距离计算。因为滑动均值训练过程中使用的中心点数量比最终得到的中心点数量稍多一些，但不会超过50%（否则要重新生成初始中心点），所以滑动均值一轮的计算量与k均值相当。裁减中心点时，每裁减一个中心点，都要训练苦干轮。考虑到减少一个中心点对聚类的影响不太特别大，训练少数几轮就可以收敛。假设初始中心点数量比最终中心点数多m个，那么滑动均值的计算量的上限是k均值计算量的$m+1$倍。

不用指定类别数量。滑动均值会根据指定的半径自行决定类别数量。

性能好于k均值。在鸢尾花数据集和手写数字数据上的测试结果已经说明了这一点。

滑动均值需要在更多的数据集上测试，以便观察其特点，进一步提高聚类效果、减少计算量。

参考文献

[1] 周志华, 机器学习, p197-220，清华大学出版社，2016.4.

[2] Chao Li, Zhiyuan Liu, et al. Multi-Interest Network with Dynamic Routing for Recommendation at Tmall. 2019.4. arXiv:1904.08030.

[3] Lloyd, Stuart P.. Least Squares Quantization in PCM. IEEE Transactions on Information Theory. Vol. 28, 1982, pp. 129 – 137.

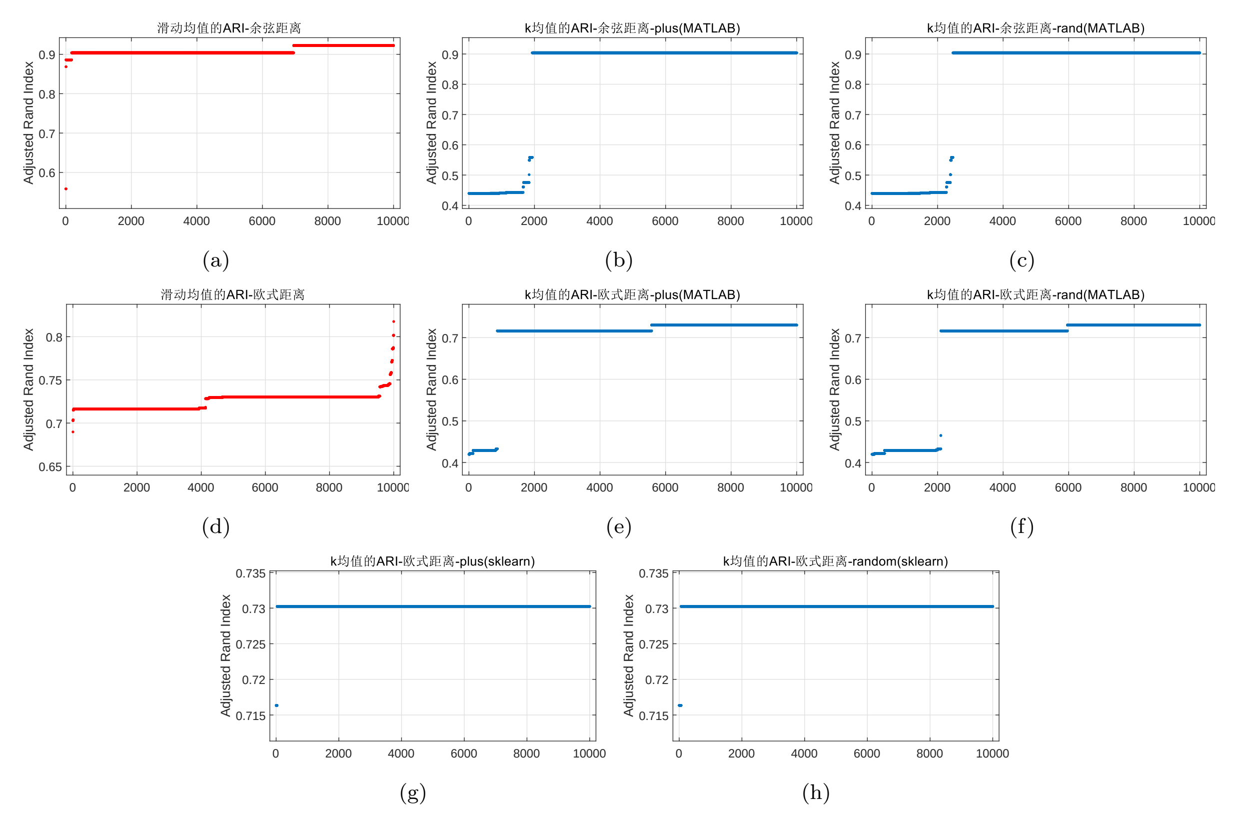


图 2: 滑动均值和k均值在iris上的表现。(a)-(c)采用余弦距离，(d)-(h)采用欧式距离。

[4] Ester, M., H. et al. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, Portland, OR, AAAI Press, pp. 226 – 231. 1996.

[5] Ankerst, Mihael, et al. OPTICS: ordering points to identify the clustering structure. In ACM Sigmod Record, vol. 28, no. 2, pp. 49-60. ACM, 1999.

[6] Arthur, David, and Sergi Vassilvitskii. K-means++: The Advantages of Careful Seeding. SODA '07: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms. 2007, pp. 1027 – 1035.

[7] McCallum, A.; Nigam, K.; and Ungar L.H. Efficient Clustering of High Dimensional Data Sets with Application to Reference Matching. Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, 169-178. 2000. doi:10.1145/347090.347123

[8] R. A. Fisher. The use of multiple measurements in taxonomic problems. Annals of Eugenics. 7 (2): 179 – 188. 1936. doi:10.1111/j.1469-1809.1936.tb02137.x. hdl:2440/15227.

[9] Dua, D. and Graff, C.. UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science. 2019.

[10] C. Kaynak. Methods of Combining Multiple Classifiers and Their Applications to Handwritten Digit Recognition, MSc Thesis, Institute of Graduate Studies in Science and Engineering, Bogazici University. 1995.

[11] L. Hubert and P. Arabie. "Comparing Partitions" Journal of Classification 2:193-218. 1985.

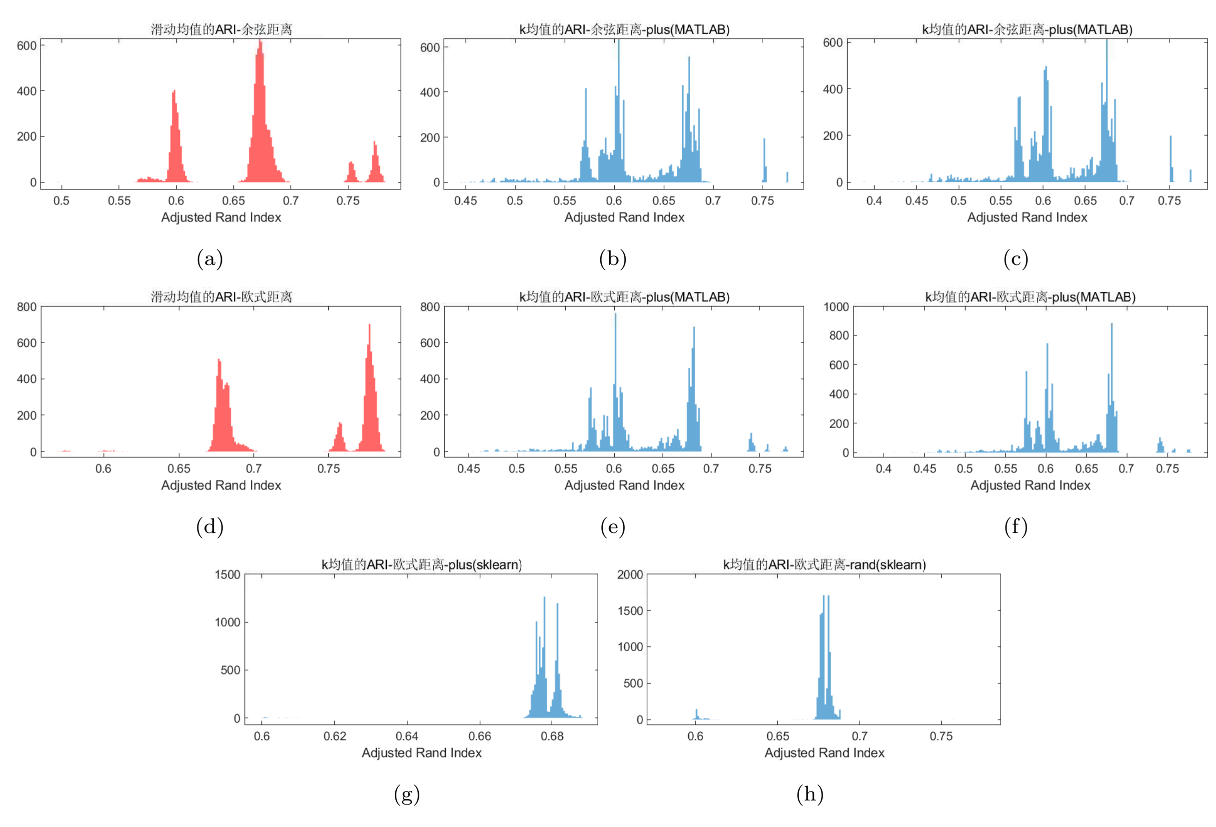


图 3: 滑动均值和k均值在手写数字上的性能。(a)-(c)采用余弦距离，(d)-(h)采用欧式距离。